SEC-PROJECT 2019 Stage 1 Report

G6 - 78093, 83420, 84740

The system is composed by a notary server (notary) which provides an API to perform operations, a client library (notary-client) which implements remote calls to the notary and abstracts those calls into a Java Object, the user server (user) that provides the user API such as the transfer good operation, and an interface for the users to perform their transactions and queries (user-client). The project also implements an abstract utility library with the cryptographic functions required for the system, which includes operations to generate, read and write asymmetric keys and functions to deal with digital signatures operations (generate, validate, etc). The servers communications are using RESTful API with Java Jersey.

To ensure **integrity** and **authenticity**, messages carry digital signatures. In every request made to the server, the parameters are concatenated and encrypted with the private key of the user that is making the request. The receiver of the message constructs the same string with the parameters received and check with the sender's public key, if the signature is valid or not. Since the message is signed with the private key of each user/notary, the authenticity of the message is also ensured. This also ensures the integrity of the message since the method, their parameters and the nonces are signed. If there is any kind of attempt to tamper the message, it will be detected in the signature verification and the message will be rejected.

To prevent **replay attacks**, Unix timestamps (up to the millisecond) are added to the messages. We chose this method as every nonce generation will be unique, and will presumably represent a bigger number than the previously generated nonces. The timestamps corresponding to the last message received from each user are stored in the notary. Thus, when the notary receives a message it verifies if the signature is valid and then verifies if the timestamp is later than the last received timestamp, ensuring freshness. The users receive the response containing the content sent concatenated with a timestamp of the notary, avoiding the replay of previous responses. This also allows the user to verify if the answer corresponds to the request. The replay attacks are mitigated with this mechanism because these are also signed by the sender, which disable content forgery. In addition, this prevents a situation where an attack window exists.
In the case of the method transferGood, the notary receives the Buyer and the Seller's signature to ensure that the Buyer was the one who wanted to obtain the good and that the Seller is the one who wants to sell it (authenticity).

To ensure **dependability** in the Notary, if there is a change in the notary data structures, a save operation will be performed (with object serialization). This will be done before communicating the response to the requester. This mechanism is expensive, since we are writing at the end of any operation but this ensures no data loss between operations.

When the Citizen Card is activated, the notary will generate ephemeral keys to provide digital signatures to operations other then transfer goods. This is done to decrease the use of the Citizen Card private key and avoid statistical attacks do the Citizen Card private key.

During BootStrap the notary generates and sign the public key with the citizen card. Then it provides the public key with Citizen Card signature through an endpoint.

All users private keys are encrypted with a password and a salt and are stored in files.